

# Gaming security by obscurity

Dusko Pavlovic

Royal Holloway, University of London, and University of Twente  
dusko.pavlovic@rhul.ac.uk

## ABSTRACT

Shannon [40] sought security against the attacker with unlimited computational powers: *if an information source conveys some information, then Shannon's attacker will surely extract that information*. Diffie and Hellman [13] refined Shannon's attacker model by taking into account the fact that the real attackers are computationally limited. This idea became one of the greatest new paradigms in computer science, and led to modern cryptography.

Shannon also sought security against the attacker with unlimited logical and observational powers, expressed through the maxim that "the enemy knows the system". This view is still endorsed in cryptography. The popular formulation, going back to Kerckhoffs [24], is that "there is no security by obscurity", meaning that the algorithms cannot be kept obscured from the attacker, and that security should only rely upon the secret keys. In fact, modern cryptography goes even further than Shannon or Kerckhoffs in tacitly assuming that *if there is an algorithm that can break the system, then the attacker will surely find that algorithm*. The attacker is not viewed as an omnipotent computer any more, but he is still construed as an omnipotent programmer. The ongoing hackers' successes seem to justify this view.

So the Diffie-Hellman step from unlimited to limited computational powers has not been extended into a step from unlimited to limited logical or programming powers. Is the assumption that all feasible algorithms will eventually be discovered and implemented really different from the assumption that everything that is computable will eventually be computed? The present paper explores some ways to refine the current models of the attacker, and of the defender, by taking into account their limited logical and programming powers. If the adaptive attacker actively queries the system to seek out its vulnerabilities, can the system gain some security by actively learning attacker's methods, and adapting to them?

## 1. INTRODUCTION

New paradigms change the world. In computer science, they often sneak behind researchers' backs: the grand visions often frazzle into minor ripples (like the fifth generation of programming languages), whereas some modest goals engender tidal waves with global repercussions (like moving the cursor by a device with wheels, or connecting remote computers). So it is not easy to conjure a new paradigm when you need it.

Perhaps the only readily available method to generate new paradigms at leisure is by disputing the obvious. Just in case, I question on this occasion not one, but two generally endorsed views:

- *Kerckhoffs Principle* that there is no security by obscurity, and
- *Fortification Principle* that the defender has to defend all attack vectors, whereas the attacker only needs to attack one.

To simplify things a little, I argue that these two principles are related. The Kerckhoffs Principle demands that a system should withstand attackers unhindered probing. In the modern security definitions, this is amplified to the requirement that the system should resist a family of attacks, irrespective of the details of their algorithms. The adaptive attackers are thus allowed to query the system, whereas the system is not allowed to query the attackers. The resulting *information asymmetry* makes security look like a game biased in favor of the attackers. The Fortification Principle is an expression of that asymmetry. In economics, information asymmetry has been recognized as a fundamental problem, worth the Nobel Prize in Economics for 2001 [44, 3, 42]. In security research, the problem does not seem to have been explicitly addressed, but there is, of course, no shortage of efforts to realize security by obscurity in practice — albeit without any discernible method. Although the practices of analyzing the attackers and hiding the systems are hardly waiting for anyone to invent a new paradigm, I will pursue the possibility that a new paradigm might be sneaking behind our backs again, like so many old paradigms did.

### Outline of the paper

While I am on the subject of security paradigms, I decided to first spell out a general overview of the old ones. An

attempt at this is in Sec. 2. It is surely incomplete, and perhaps wrongheaded, but it may help a little. It is difficult to communicate about the new without an agreement about the old. Moreover, it will be interesting to hear not only whether my new paradigms are new, but also whether my old paradigms are old.

The new security paradigm arising from the slogan "*Know your enemy*" is discussed in Sec. 3. Of course, security engineers often know their enemies, so this is not much of a new paradigm in practice. But security researchers often require that systems should be secure against universal families of attackers, without knowing anything about who the enemy is at any particular moment. With respect to such static requirements, a game theoretic analysis of dynamics of security can be viewed as an almost-new paradigm (with few previous owners). In Sec.3.1 I point to the practical developments that lead up to this paradigm, and then in Sec. 3.2 I describe the game of attack vectors, which illustrates it. This is a very crude view of security process as a game of incomplete information. I provide a simple pictorial analysis of the strategic interactions in this game, which turn out to be based on acquiring information about the opponent's type and behavior. A sketch of a formal model of security games of incomplete information, and of the game of attack vectors, is given in Appendix A.

A brand new security paradigm of "*Applied security by obscurity*" is described in Sec. 4. It is based on the idea of *logical complexity* of programs, which leads to one way programming similarly like computational complexity led to one way computations. If achieved, one way programming will be a powerful tool in security games.

A final attempt at a summary, and some comments about the future research, and the pitfalls, are given in Sec. 5.

### Related work

The two new paradigms offer two new tools for the security toolkit: games of incomplete information, and algorithmic information theory.

Game theoretic techniques have been used in applied security for a long time, since there a need for strategic reasoning often arises in practice. A typical example from the early days is [12], where games of imperfect information were used. Perhaps the simplest more recent game based model are the attack-defense trees, which boil down to zero-sum extensive games [27]. Another application of games of imperfect information appeared, e.g., in a previous edition of this conference [31]. Conspicuously, games of incomplete information do not seem to have been used, which seems appropriate since they analyze how players keep each other in obscurity. The coalgebraic presentation of games and response relations, presented in the Appendix, is closely related with the formalism used in [36].

The concept of logical complexity, proposed in Sec. 4, is based on the ideas of algorithmic information theory [26, 41] in general, and in particular on the idea of *logical depth* [11, 28, 7]. I propose to formalize logical complexity by lifting logical depth from the Gödel-Kleene indices to program

specifications [21, 9, 10, 15, 39, 37]. The underlying idea that a Gödel-Kleene index of a program can be viewed as its "explanation" goes back to Kleene's idea of *realizability* [25] and to Solomonoff's formalization of inductive inference [41].

## 2. OLD SECURITY PARADIGMS

Security means many things to many people. For a software engineer, it often means that there are no buffer overflows or dangling pointers in the code. For a cryptographer, it means that any successful attack on the cypher can be reduced to an algorithm for computing discrete logarithms, or to integer factorization. For a diplomat, security means that the enemy cannot read the confidential messages. For a credit card operator, it means that the total costs of the fraudulent transactions and of the measures to prevent them are low, relative to the revenue. For a bee, security means that no intruder into the beehive will escape her sting...

Is it an accident that all these different ideas go under the same name? What do they really have in common? They are studied in different sciences, ranging from computer science to biology, by a wide variety of different methods. Would it be useful to study them together?

### 2.1 What is security?

If all avatars of security have one thing in common, it is surely the idea that *there are enemies and potential attackers out there*. All security concerns, from computation to politics and biology, come down to averting the adversarial processes in the environment, that are poised to subvert the goals of the system. There are, for instance, many kinds of bugs in software, but only those that the hackers use are a security concern.

In all engineering disciplines, the system guarantees a functionality, provided that the environment satisfies some assumptions. This is the standard *assume-guarantee* format of the engineering correctness statements. Such statements are useful when the environment is passive, so that the assumptions about it remain valid for a while. The essence of security engineering is that the environment actively seeks to invalidate system's assumptions.

Security is thus an *adversarial process*. In all engineering disciplines, failures usually arise from engineering errors and noncompliance. In security, failures arise *in spite* of the compliance with the best engineering practices of the moment. Failures are the first class citizens of security: every key has a lifetime, and in a sense, every system too. For all major software systems, we normally expect security updates, which usually arise from attacks, and often inspire them.

### 2.2 Where did security come from?

The earliest examples of security technologies are found among the earliest documents of civilization. Fig. 1 shows security tokens with a tamper protection technology from almost 6000 years ago. Fig.2 depicts the situation where this technology was probably used. Alice has a lamb and Bob has built a secure vault, perhaps with multiple security levels,



Figure 1: Tamper protection from 3700 BC

spacious enough to store both Bob's and Alice's assets. For each of Alice's assets deposited in the vault, Bob issues a clay token, with an inscription identifying the asset. Alice's tokens are then encased into a *bullā*, a round, hollow "envelope" of clay, which is then baked to prevent tampering. When she wants to withdraw her deposits, Alice submits her bulla to Bob, he breaks it, extracts the tokens, and returns the goods. Alice can also give her bulla to Carol, who can also submit it to Bob, to withdraw the goods, or pass on to Dave. Bullae can thus be traded, and they facilitate exchange economy. The tokens used in the bullae evolved into the earliest forms of money, and the inscriptions on them led to the earliest numeral systems, as well as to Sumerian cuneiform script, which was one of the earliest alphabets. Security thus predates literature, science, mathematics, and even money.

### 2.3 Where is security going?

Through history, security technologies evolved gradually, serving the purposes of war and peace, protecting public resources and private property. As computers pervaded all aspects of social life, security became interlaced with computation, and security engineering came to be closely related with computer science. The developments in the realm of security are nowadays inseparable from the developments in the realm of computation. The most notable such development is, of course, *cyber space*.

#### *Paradigms of computation*

In the beginning, engineers built computers, and wrote programs to control computations. The platform of computation was the computer, and it was used to execute algorithms and calculations, allowing people to discover, e.g., fractals, and to invent compilers, that allowed them to write and execute more algorithms and more calculations more efficiently. Then the operating system became the platform of computation, and software was developed on top of it. The era of personal computing and enterprise software broke out. And then the Internet happened, followed by cellular networks, and wireless networks, and ad hoc networks, and mixed networks. Cyber space emerged as the distance-free space of instant, costless communication. Nowadays software is developed to run in cyberspace. The Web is, strictly speaking, just a software system, albeit a formidable one. A botnet is

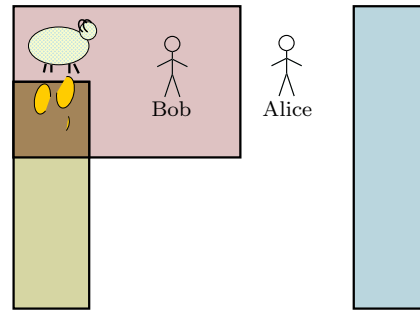


Figure 2: To withdraw her sheep from Bob's secure vault, Alice submits a tamper-proof token from Fig. 1.

also a software system. As social space blends with cyber space, many social (business, collaborative) processes can be usefully construed as software systems, that ran on social networks as hardware. Many social and computational processes become inextricable. Table 1 summarizes the crude picture of the paradigm shifts which led to this remarkable situation.

But as every person got connected to a computer, and every computer to a network, and every network to a network of networks, computation became interlaced with communication, and ceased to be programmable. The functioning of the Web and of web applications is not determined by the code in the same sense as in a traditional software system: after all, web applications do include the human users as a part of their runtime. The fusion of social and computational processes in cyber-social space leads to a new type of information processing, where the purposeful program executions at the network nodes are supplemented by spontaneous data-driven evolution of network links. While the network emerges as the new computer, data and metadata become inseparable, and a new type of security problems arises.

#### *Paradigms of security*

In early computer systems, security tasks mainly concerned sharing of the computing resources. In computer networks, security goals expanded to include information protection. Both computer security and information security essentially depend on a clear distinction between the secure areas, and the insecure areas, separated by a security perimeter. Security engineering caters for computer security and for information security by providing the tools to build the security perimeter. In cyber space, the secure areas are separated from the insecure areas by the "walls" of cryptography; and they are connected by the "gates" of cryptographic protocols.<sup>1</sup> But as networks of computers and devices spread through physical and social spaces, the distinctions between

<sup>1</sup>This is, of course, a blatant oversimplification, as are many other statements I make. In a sense, every statement is an oversimplification of reality, abstracting away the matters deemed irrelevant. The gentle reader is invited to squint whenever any of the details that I omit do seem relevant, and add them to the picture. The shape of a forest should not change when some trees are enhanced.

<i>age</i>	<i>ancient times</i>	<i>middle ages</i>	<i>modern times</i>
<b>platform</b>	computer	operating system	network
<b>applications</b>	Quicksort, compilers	MS Word, Oracle	WWW, botnets
<b>requirements</b>	correctness, termination	liveness, safety	trust, privacy
<b>tools</b>	programming languages	specification languages	scripting languages

**Table 1: Paradigm shifts in computation**

<i>age</i>	<i>middle ages</i>	<i>modern times</i>	<i>postmodern times</i>
<b>space</b>	computer center	cyber space	cyber-social space
<b>assets</b>	computing resources	information	public and private resources
<b>requirements</b>	availability, authorization	integrity, confidentiality	trust, privacy
<b>tools</b>	locks, tokens, passwords	cryptography, protocols	mining and classification

**Table 2: Paradigm shifts in security**

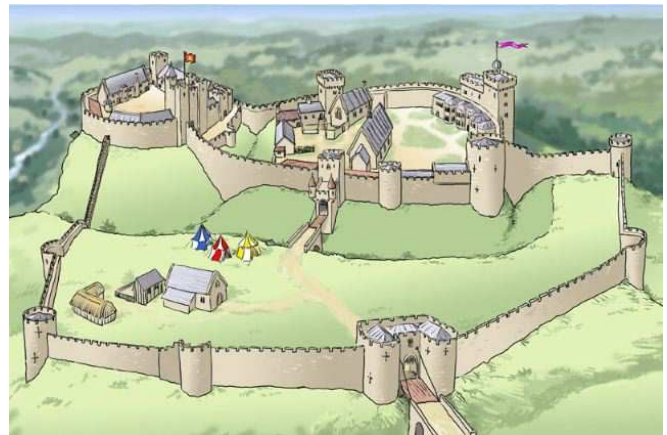
the secure and the insecure areas become blurred. And in such areas of cyber-social space, information processing does not yield to programming, and cannot be secured just by cryptography and protocols. What else is there?

### 3. A SECOND-HAND BUT ALMOST-NEW SECURITY PARADIGM: KNOW YOUR ENEMY

#### 3.1 Security beyond architecture

Let us take a closer look at the paradigm shift to postmodern cyber security in Table 2. It can be illustrated as the shift from Fig. 3 to Fig. 4. The fortification in Fig. 3 represents the view that security is in essence an architectural task. A fortress consists of walls and gates, separating the secure area within from the insecure area outside. The boundary between these two areas is the security perimeter. The secure area may be further subdivided into the areas of higher security and the areas of lower security. In cyber space, as we mentioned, the walls are realized using crypto systems, whereas the gates are authentication protocols. But as every fortress owner knows, the walls and the gates are not enough for security: you also need some soldiers to defend it, and some weapons to arm the soldiers, and some craftsmen to build the weapons, and so on. Moreover, you also need police and judges to maintain security within the fortress. They take care for the dynamic aspects of security. These dynamic aspects arise from the fact that sooner or later, the enemies will emerge inside the fortress: they will scale the walls at night (i.e. break the crypto), or sneak past the gatekeepers (break the protocols), or build up trust and enter with honest intentions, and later defect to the enemy; or enter as moles, with the intention to strike later. In any case, security is not localized at the security perimeters of Fig. 3, but evolves in-depth, like on Fig. 4, through social processes, like trust, privacy, reputation, influence.

In summary, besides the methods to keep the attackers out, security is also concerned with the methods to deal with



**Figure 3: Static security**



**Figure 4: Dynamic security**

the attackers once they get in. Security researchers have traditionally devoted more attention to the former family of methods. Insider threats have attracted a lot of attention recently, but a coherent set of research methods is yet to emerge.

Interestingly, though, there is a sense in which security becomes an easier task when the attacker is in. Although unintuitive at the first sight, this idea becomes natural when security processes are viewed in a broad context of the information flows surrounding them (and not only with respect to the data designated to be secret or private). To view security processes in this broad context, it is convenient to model them as *games of incomplete information* [4], where the players do not have enough information to predict the opponent's behavior. For the moment, let me just say that the two families of security methods (those to keep the attackers out, and those to catch them when they are in) correspond to two families of strategies in certain games of incomplete information, and turn out to have quite different winning odds for the attacker, and for defender. In fact, they have the *opposite* winning odds.

In the fortress mode, when the defenders' goal is to keep the attackers out, it is often observed that the attackers only need to find one attack vector to enter the fortress, whereas the defenders must defend all attack vectors to prevent them. When the battle switches to the dynamic mode, and the defense moves inside, then the defenders only need to find one marker to recognize and catch the attackers, whereas the attackers must cover all their markers. This strategic advantage is also the critical aspect of the immune response, where the invading organisms are purposely sampled and analyzed for chemical markers. Some aspects of this observation have, of course, been discussed within the framework of biologically inspired security. Game theoretic modeling seems to be opening up a new dimension in this problem space. We present a sketch to illustrate this new technical and conceptual direction.

## 3.2 The game of attack vectors

**Arena.** Two players, the attacker  $A$  and the defender  $D$ , battle for some assets of value to both of them. They are given equal, disjoint territories, with the borders of equal length, and equal amounts of force, expressed as two vector fields distributed along their respective borders. The players can redistribute the forces and move the borders of their territories. The territories can thus take any shapes and occupy any areas where the players may move them, obeying the constraints that

- (i) the length of the borders of both territories must be preserved, and
- (ii) the two territories must remain disjoint, except that they may touch at the borders.

It is assumed that the desired asset  $\Theta$  is initially held by the defender  $D$ . Suppose that storing this asset takes an area of size  $\theta$ . Defender's goal is thus to maintain a territory  $p_D$  with an area  $\int p_D \geq \theta$ . Attacker's goal is to decrease the

size of  $p_D$  below  $\theta$ , so that the defender must release some of the asset  $\Theta$ . To achieve this, the attacker  $A$  must bring his<sup>2</sup> forces to defender  $D$ 's borders, and push into his territory. A position in the game can thus be something like Fig. 5.

**Game.** At each step in the game, each player makes a move by specifying a distribution of his forces along his borders. Both players are assumed to be able to redistribute their forces with equal agility. The new force vectors meet at the border, they add up, and the border moves along the resulting vector. So if the vectors are, say, in the opposite directions, the forces subtract and the border is pushed by the greater vector.

The players observe each other's positions and moves in two ways:

- (a) Each player knows his own moves, i.e. distributions, and sees how his borders change. From the change in the previous move, he can thus derive the opponent's current distribution of the forces along the common part of the border.
- (b) Each player sees all movement in the areas enclosed enclosed within his territory, i.e. observes any point on a straight line between any two points that he controls. That means that each player sees the opponent's next move at all points that lie within the convex hull of his territory, which we call *range*.

According to (b), the position in Fig. 5 allows  $A$  to see  $D$ 's next move.  $D$ , on the other hand only gets to know  $A$ 's move according to (a), when his own border changes. This depicts, albeit very crudely, the information asymmetry between the attacker and the defender.

**Question.** *How should rational players play this game?*

### 3.2.1 Fortification strategy

**Goals.** Since each player's total force is divided by the length of his borders, the maximal area defensible by a given force has the shape of a disk. All other shapes with the boundaries of the same length enclose smaller areas. So  $D$ 's simplest strategy is to acquire and maintain the smallest disk shaped territory of size  $\theta$ .<sup>3</sup> This is the *fortification* strategy:  $D$  only responds to  $A$ 's moves.

$A$ 's goal is, on the other hand, to create "dents" in  $D$ 's territory  $p_D$ , since the area of  $p_D$  decreases most when its convexity is disturbed. If a dent grows deep enough to reach across  $p_D$ , or if two dents in it meet, then  $p_D$  disconnects

<sup>2</sup>I hope no one minds that I will be using "he" for both Attacker and Defender, in an attempt to avoid distracting connotations.

<sup>3</sup>This is why the core of a medieval fortification was a round tower with a thick wall and a small space inside. The fortress itself often is not round, because the environment is not flat, or because the straight walls were easier to build; but it is usually at least convex. Later fortresses, however, had protruding towers — to attack the attacker. Which leads us beyond the fortification strategy...



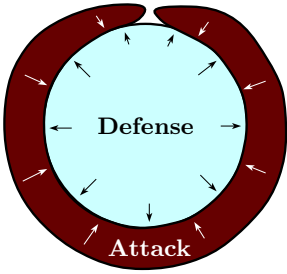


Figure 5: Fortification

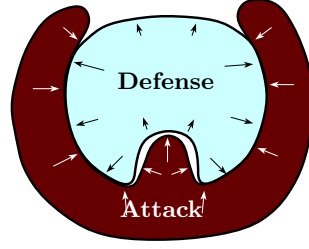


Figure 6: Honeypot

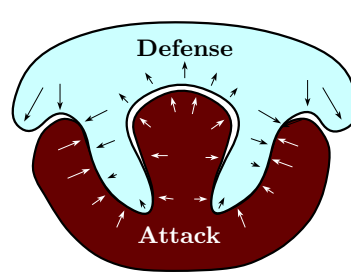


Figure 7: Sampling

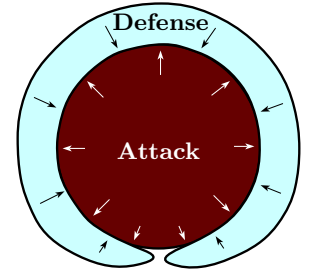


Figure 8: Adaptation

in two components. Given a constant length of the border, it is easy to see that the size of the enclosed area decreases exponentially as it gets broken up. In this way, the area enclosed by a border of given length can be made arbitrarily small.

But how can  $A$  create dents? Wherever he pushes, the defender will push back. Since their forces are equal and constant, increasing the force along one vector decreases the force along another vector.

**Optimization tasks.** To follow the fortification strategy,  $D$  just keeps restoring  $p_D$  to a disk of size  $\theta$ . To counter  $D$ 's defenses,  $A$  needs to find out where they are the weakest. He can observe this wherever  $D$ 's territory  $p_D$  is within  $A$ 's range, i.e. contained in the convex hull of  $p_A$ . So  $A$  needs to maximize the intersection of his range with  $D$ 's territory. Fig. 5 depicts a position where this is achieved:  $D$  is under  $A$ 's siege. It embodies the Fortification Principle, that the defender must defend all attack vectors, whereas the attacker only needs to select one. For a fast push,  $A$  randomly selects an attack vector, and waits for  $D$  to push back. Strengthening  $D$ 's defense along one vector weakens it along another one. Since all of  $D$ 's territory is within  $A$ 's range,  $A$  sees where  $D$ 's defense is the weakest, and launches the next attack there. In contrast,  $D$ 's range is initially limited to his own disk shaped territory. So  $D$  only "feels"  $A$ 's pushes when his own borders move. At each step,  $A$  pushes at  $D$ 's weakest point, and creates a deeper dent.  $A$  does enter into  $D$ 's range, but  $D$ 's fortification strategy makes no use of the information that could be obtained about  $A$ . The number of steps needed to decrease  $p_D$  below  $\theta$  depends on how big are the forces and how small are the contested areas.

### 3.2.2 Adaptation strategy

What can  $D$  do to avoid the unfavorable outcome of the fortification strategy? The idea is that he should learn to know his enemy: he should also try to shape his territory to maximize the intersection of his range with  $A$ 's territory.  $D$  can lure  $A$  into his range simply letting  $A$  dent his territory. This is the familiar *honeypot* approach, illustrated on Fig. 6. Instead of racing around the border to push back against every attack,  $D$  now gathers information about  $A$ 's next moves within his range. For this, he sacrifices a little bit of his territory, as a bait (the "honey") for  $A$ . If  $A$  sticks with his strategic preferences, he will accept  $D$ 's sacrifice, and enter into  $D$ 's range more and more. Fig. 7 depicts a

further step in  $D$ 's strategic development, where he does not just passively wait for  $A$  to enter the honeypot, but actively herds  $A$  into it. This is the *sampling* strategy. Formally, it is characterized by  $D$ 's higher valuation for the information about  $A$ , than for the territory alone. This is reflected in the fact that  $D$ 's territory gradually evolves into a shape optimized for information gathering. In the cyber wars of the day, the sampling strategy is emerging, e.g., among the researchers who have gone beyond luring some bots into some sandboxed computers, and hijacked parts of botnets from their owners, for the sole purpose of research [46, 45]. Continuing in this direction leads to the long term strategy of *adaptation*, depicted on Fig. 8, where all of  $D$ 's strategic valuation is assigned to the information about the opponent. Here  $A$ 's attacks are actively observed and prevented; the territory is maintained as a side effect of keeping the opponent localized. In the long term,  $D$  wins. The proviso is that  $D$  has enough territory to begin with. A simplifying assumption of the presented model is that  $A$  blindly sticks with his valuation of the territory, leading him to accept all baits. Reality is, of course, not so simple, and  $A$ 's strategy also allows various refinements. However, to achieve his goal of stealing  $D$ 's assets,  $A$  cannot avoid entering  $D$ 's range altogether.  $D$ , on the other hand, cannot allow that the size of his territory drops below  $\theta$ . Respecting these asymmetric constraints, both players' strategy refinements will evolve methods to trade territory for information, making increasingly efficient use of both.

A formalism for a mathematical analysis of this game is sketched in the Appendix.

## 3.3 What does all this mean for security?

The presented toy model provides a very crude picture of the evolution of defense strategies from fortification to adaptation. Intuitively, Fig. 5 can be viewed as a fortress under siege, whereas Fig. 8 can be interpreted as a macrophage localizing an invader. The intermediate pictures show the adaptive immune system luring the invader and sampling his chemical markers.

But there is nothing exclusively biological about the adaptation strategy. Figures 5–8 could also be viewed entirely in the context of Figures 3–4, and interpreted as the transition from the medieval defense strategies to modern political ideas. Fig. 8 could be viewed as a depiction of the idea of "*preemptive siege*": while the medieval rulers tried to keep

their enemies out of their fortresses, some of the modern ones try to keep them in their jails. The evolution of strategic thinking illustrated on Figures 5-8 is pervasive in all realms of security, i.e. wherever the adversarial behaviors are a problem, including cyber-security.

And although the paradigm of keeping an eye on your enemies is familiar, the fact that it reverts the odds of security and turns them in favor of the defenders does not seem to have received enough attention. It opens up a new game theoretic perspective on security, and suggests a new tool for it.

## 4. A BRAND NEW SECURITY PARADIGM: APPLIED SECURITY BY OBSCURITY

### 4.1 Gaming security basics

**Games of information.** In games of luck, each player has a *type*, and some *secrets*. The type determines player's preferences and behaviors. The secrets determine player's state. E.g., in poker, the secrets are the cards in player's hand, whereas her type consists of her risk aversion, her gaming habits etc. The *imperfect* information means that all players' types are a public information, whereas their states are unknown, because their secrets are private. In games of *incomplete* information, both players' types and their secrets are unknown. The basic ideas and definitions of the complete and incomplete informations in games go all the way back to von Neumann and Morgenstern [48]. The ideas and techniques for modeling incomplete information are due to Harsanyi [20], and constitute an important part of game theory [30, 18, 4].

**Security by secrecy.** If cryptanalysis is viewed as a game, then the algorithms used in a crypto system can be viewed as the type of the corresponding player. The keys are, of course, its secrets. In this framework, Claude Shannon's slogan that "*the enemy knows the system*" asserts that cryptanalysis should be viewed as a game of imperfect information. Since the type of the crypto system is known to the enemy, it is not a game of incomplete information. Another statement of the same imperative is the Kerckhoffs' slogan that "*there is no security by obscurity*". Here the obscurity refers to the type of the system, so the slogan thus suggests that the security of a crypto system should only depend on the secrecy of its keys, and remain secure if its type is known. In terms of physical security, both slogans thus say that the thief should not be able to get into the house without the right key, even if he knows the mechanics of the lock. The key is the secret, the lock is the type.

**Security by obscurity.** And while all seems clear, and we all pledge allegiance to the Kerckhoffs' Principle, the practices of security by obscurity abound. E.g., besides the locks that keep the thieves out, many of us use some child-proof locks, to protect toddlers from dangers. A child-proof lock usually does not have a key, and only provides protection through the obscurity of its mechanism.

On the cryptographic side, security by obscurity remains one of the main tools, e.g., in Digital Rights Management

(DRM), where the task is to protect the digital content from its intended users. So our DVDs are encrypted to prevent copying; but the key must be on each DVD, or else the DVD could not be played. In order to break the copy protection, the attacker just needs to find out where to look for the key; i.e. he needs to know the system used to hide the key. For a sophisticated attacker, this is no problem; but the majority is not sophisticated. The DRM is thus based on the second-hand but almost-new paradigm from the preceding section: the DVD designers study the DVD users and hide the keys in obscure places. From time to time, the obscurity wears out, by an advance in reverse engineering, or by a lapse of defenders attention<sup>4</sup>. Security is then restored by analyzing the enemy, and either introducing new features to stall the ripping software, or by dragging the software distributors to court. Security by obscurity is an ongoing process, just like all of security.

### 4.2 Logical complexity

**What is the difference between keys and locks?** The conceptual problem with the Kerckhoffs Principle, as the requirement that security should be based on secret keys, and not on obscure algorithms, is that it seems inconsistent, at least at the first sight, with the Von Neumann architecture of our computers, where programs are represented as data. In a computer, both a key and an algorithm is a string of bits. Why can I hide a key and cannot hide an algorithm? More generally, why can I hide data, and cannot hide programs?

Technically, the answer boils down to the difference between data encryption and program obfuscation. The task of encryption is to transform a data representation in such a way that it can be recovered if and only if you have a key. The task of obfuscation is to transform a program representation so that the obfuscated program runs roughly the same as the original one, but that the original code (or some secrets built into it) cannot be recovered. Of course, the latter is harder, because encrypted data just need to be secret, whereas an obfuscated program needs to be secret *and* to run like the original program. In [5], it was shown that some programs must disclose the original code in order to perform the same function (*and* they disclose it in a nontrivial way, i.e. not by simply printing out their own code). The theory here confirms the empiric evidence that reverse engineering is, on the average<sup>5</sup>, effective enough that you don't want to rely upon its hardness. So it is much easier to find out the lock mechanism, than to find the right key, even in the digital domain. When they say that there is no security by obscurity, security practitioners thus usually mean that re-

<sup>4</sup>The DVD Copy Scramble System (CSS) was originally reverse engineered to allow playing DVDs on Linux computers. This was possibly facilitated by an inadvertent disclosure from the DVD Copy Control Association (CAA). DVD CAA pursued the authors and distributors of the Linux DeCSS module through a series of court cases, until the case was dismissed in 2004 [16]. Ironically, the cryptography used in DVD CSS has been so weak, in part due to the US export controls at the time of design, that any computer fast enough to play DVDs could find the key by brute force within 18 seconds [43]. This easy cryptanalytic attack was published before DeCSS, but seemed too obscure for everyday use.

<sup>5</sup>However, the *International Obfuscated C Code Contest* [23] has generated some interesting and extremely amusing work.

verse engineering is easy, whereas cryptanalysis is hard, and provides more durable security guarantees.<sup>6</sup>

**One-way programming?** Modern cryptography is based on one way functions, which are easy to compute, but hard to invert. Secure systems are designed to use the easy direction of one-way functions, whereas the attacks must invert them, i.e. compute the hard direction. A high level view thus displays security as a process where the attackers program attack algorithms in response to the algorithms of the system that they attack, whereas the defenders program system algorithms in response to some attacks. The question is now whether systems can be designed in such a way to make the defenders' programming tasks easy, and the attackers' programming tasks hard. Can we lift the idea of one-way functions to one-way programming?

Let us take a closer look. How can we make attacker's task harder? Since obfuscation is hard and reverse engineering is easy, we assume that the system code is accessible to the attacker, and that the attack code is accessible to the defender. This assumption is supported by the current security practices, where the attacker communities reverse engineer their target systems, and the security researchers decompile malware. But even when the code is completely transparent, the attacker's task of uncovering vulnerabilities of the system remains nontrivial. And even with a trove of detected vulnerabilities, the attacker still needs to design an effective attack algorithm to exploit them. The hard part of an attacker's job is thus the *logical* task to analyze the system, and to design and implement an attack algorithm. The *logical complexity* of such tasks is different from the *computational complexity* of the system and the attack algorithms. Indeed, a computationally easy algorithm may be logically hard to construct, even when it can be expressed by a relatively succinct program, like e.g. [2]; whereas an algorithm that requires a minor logical effort to construct may, of course, require a great computational effort to run.

The different roles of the computational and the logical complexities in security can perhaps be pondered on the following example. In modern cryptography, a system  $C$  would be considered very secure if an attack algorithm  $A_C$  on it would yield a proof that  $P = NP$ . But how would you feel about a crypto system  $L$  such that an attack algorithm  $A_L$  would yield a proof that  $P \neq NP$ ? What is the difference between the reductions

$$A_C \implies P = NP \quad \text{and} \quad A_L \implies P \neq NP \quad ?$$

The security of the system  $C$  is based on the *computational* complexity of the  $NP$  problems. The security of the system  $L$  is based on the *logical* complexity of proving  $P \neq NP$ . Most computer scientists believe that  $P \neq NP$  is true. If  $P = NP$  is thus false, then no attack on the system  $C$  can exist, whereas an attack on the system  $L$  may very well be possible. So the security of the system  $L$  may very well be based on the *obscurity* of the proof of  $P \neq NP$ , which most likely exists, but is very hard to find. The best minds of mankind have spent many years looking for this proof, but did not

<sup>6</sup>It is interesting to note that one of the initial ideas for public key crypto system, suggested in [13], was to partially evaluate a symmetric encryption module over its key, and to publish its obfuscation as a public encryption module.

manage to find it. Yet an attack  $A_L$ , together with the security reduction  $A_L \implies P \neq NP$ , would provide such a proof. This attack would probably be welcomed with admiration and gratitude. In fact, the system  $L$  is secure enough to protect a bank account with a \$1,000,000, since proving (or disproving)  $P \neq NP$  is worth a Clay Institute Millennium Prize of \$ 1,000,000. If an attacker takes your money from the bank account, he will leave you with a proof worth much more. So the logical complexity of the system  $L$  provides enough obscurity for a significant amount of security!

**But what is logical complexity?** Computational complexity of a program tells how many computational steps (counting them in time, memory, state changes, etc.) does the program take to transform its input into its output. Logical complexity is not concerned with the execution of the program, but with its logical construction. Intuitively, if computational complexity of a program counts the number of computational steps needed to execute it on an input of a given length, its logical complexity should count the number of computational steps needed to derive that program from some given programming knowledge. For instance, while the computational complexity of an attack on a crypto system is the the number of computational steps that it requires to extract some information about the plaintext from the cyphertext, the logical complexity of that attack is the number of logical steps needed to find that attack algorithm from a given description of the system algorithm. In other words, the logical complexity of an attack on a crypto system is the computational complexity of the task of finding a counterexample for the security claim of that system.<sup>7</sup>

The idea is thus to define logical complexity of an algorithm  $A$  as the computational complexity of the fastest construction algorithm  $P_A$  that outputs  $A$ , given some algorithmic knowledge as the input. The problem in formalizing this is that the fastest program that outputs  $A$  is the program `print A`, which does not need any input, since  $A$  is hardwired in it. To assure that  $A$  is constructed in a nontrivial way, we need to look for a constructor  $P_A$  that is the fastest among the algorithms that can be implemented by programs significantly *shorter* than  $A$  itself.

This brings us into the realm of algorithmic information theory [11, 28, 29], where similar concepts, with slight variations, have been proposed under a variety of different names. Bennett's *logical depth* [7] seems to be the the closest. In Bennett's original formulation, logical depth is defined as a complexity measure assigned to data, or to observations, although Bennett's fascinating analyses also assign logical depth to genes and to organisms, as the time that it took them to evolve [6]. The step from logical depth of data to logical complexity of algorithms boils down to the view of algorithms-as-programs-as-data, originating from Gödel's enumeration of recursive functions as numbers, and from Kleene's recursive indices. Towards a brief, somewhat oversimplified, but hopefully not misleading account of these formalisms, consider the partial algebraic theory with two

<sup>7</sup>Such claims are usually stated in the form: "For all probabilistic polynomial-time Turing machines that the attacker may use, his advantage is not greater than [some formula]". The logical complexity of the attack is the computational complexity of attacker's task of finding that attack.



sorts,

- $\mathbb{N}$ , representing the data, say as numbers, and
- $\mathbb{M}$ , representing a family of algorithms, viewed as partial functions  $\mathbb{N}^* \rightarrow \mathbb{N}$ , say those that can be realized by Turing machines<sup>8</sup>,

given together with the mappings

$$\begin{array}{ccc} & \lceil - \rceil & \\ \mathbb{M} & \xleftrightarrow{\quad} & \mathbb{N} \\ & \{-\} & \end{array} \quad (1)$$

which make them isomorphic, i.e.

$$\{\lceil M \rceil\} = \quad \text{and} \quad \lceil \{n\} \rceil = n$$

This means not just that every machine  $M \in \mathbb{M}$  can be encoded as a number  $\lceil M \rceil \in \mathbb{N}$ , but also that any number  $n \in \mathbb{N}$  can be viewed as a program corresponding to the machine  $\{n\} \in \mathbb{M}$ , and executed on data. In addition, there are also

- a *universal composition machine*  $U \in \mathbb{M}$ , such that for all  $P, Q \in \mathbb{M}$  and all  $x \in \mathbb{N}$

$$U(\lceil P \rceil, \lceil Q \rceil, n) = P(Q(n)) \quad (2)$$

holds whenever either side is defined,

- a *time counter*  $T \in \mathbb{M}$ , where  $T(\lceil M \rceil, n)$  denotes the number of steps that the machine  $M$  takes before it halts on the input  $n$ , and
- a *length function*  $\ell : \mathbb{N} \rightarrow \mathbb{N}$ , assigning a to each piece of data its length.

Towards formalizing the above idea of logical complexity, we now define the *algorithmic distance*  $\mathcal{C}(A, B)$  between the algorithms  $A, B \in \mathbb{M}$  to be the length of the shortest program  $p$  that inputs the code  $\lceil A \rceil$  and outputs the code  $\lceil B \rceil$ :

$$\mathcal{C}(A, B) = \bigwedge_{\{p\}(\lceil A \rceil) = \lceil B \rceil} \ell(p) \quad (3)$$

The *logical distance*  $\mathcal{D}(A, B)$  is now the shortest time that it takes to compute  $\lceil B \rceil$  from  $\lceil A \rceil$  by one of the shortest programs:

$$\mathcal{D}(A, B) = \bigwedge_{\substack{\{p\}(\lceil A \rceil) = \lceil B \rceil \\ \ell(p) = \mathcal{C}(A, B)}} T(p, \lceil A \rceil) \quad (4)$$

**Remarks.** Algorithmic distance is based on the relativized version of Solomonoff's and Kolmogorov's definitions of complexity [41, 26]. Logical distance is based on a relativized and simplified version of Bennett's logical depth [7]. It is simplified in the sense that it does not take into account the possibility that slightly longer programs may run significantly faster. To capture this, the developed definitions of

<sup>8</sup>More precisely, they are realized as *self-delimiting* Turing machines, which allows them to receive multiple arguments on the same tape [49].

logical depth are parametrized over the difference in length between the fastest and the shortest programs. In some applications, this is an important technical detail, assuring the stability of the definition. In the current presentation, which is mainly conceptual, it would just complicate the definition. A *conceptual* detail which is also omitted is that the universal composition machine, and the programs in (3) and (4) need to be homomorphisms with respect to certain logical operations on programs. This additional requirement seems essential for the envisioned applications of logical distance as a tool of security by obscurity. Nevertheless, such matters must be left for future work. The path towards program and specification frameworks that would take into account the logical distances of algorithms, and advance the idea of one-programming, requires examining the diverse refinements of the basic idea of the Gödel-Kleene program encodings that in the meantime emerged from the theory and the extensive experience of program development [21, 10, 15, 39, 32, 38].

**Logical security.** The idea of logical security is to make the derivation of an attack algorithm from a system algorithm logically complex. Formally, a system  $S$  would thus be logically security if the distance  $\mathcal{D}(S, A)$  is large for all attacks  $A$  on  $S$ . On the other hand, since the distance is obviously not symmetric,  $\mathcal{D}(A, S)$  may be small, meaning that it may be easy to derive an improved system algorithm  $S$  from an undesired algorithm  $A$ . This connects logical security with the idea of one-way programming.

At this point, the reader may object that it does not make much sense to call  $\mathcal{D}$  a distance when  $\mathcal{D}(A, S)$  is generally different from  $\mathcal{D}(S, A)$ . A possible answer to appeal to the spatial intuition that  $S$  lays lower than  $A$ , so that getting from  $S$  to  $A$  requires strenuous climbing, and from  $A$  to  $S$  only breaking. But it gets worse. While algorithmic distance satisfies the triangle law

$$\mathcal{C}(P, Q) + \mathcal{C}(Q, R) \geq \mathcal{C}(P, R) \quad (5)$$

realized by the universal composition machine, logical distance generally does not satisfy this law, since there may be a short but slow algorithm to construct  $R$  from  $P$ , but it does not have to go through  $Q$ , and all algorithms to construct  $Q$  from  $P$  and  $R$  from  $Q$  may be long but fast. Nevertheless, logical distance can be easily shown to satisfy the weaker law

$$\mathcal{D}(P, Q) + \mathcal{D}(P \wedge Q, R) \geq \mathcal{D}(P, R) \quad (6)$$

where  $P \wedge Q$  denotes the parallel composition of  $P$  and  $Q$ , i.e. an algorithm that satisfies  $P \wedge Q(x, y) = (u, v)$  if and only if  $P(x) = u$  and  $Q(x) = v$ , for all  $x, y, u, v$ . One is tempted to call this law "*subbayesian*", since it echoes the Bayes' law for conditional probabilities in the form

$$\Pr(r \mid p) = \Pr(r \mid q \wedge p) \cdot \Pr(q \mid p)$$

where  $p, q, r$  denote events. At any rate, we can now to use (6) to bound the logical complexity  $\mathcal{D}(S, A)$  of constructing an attack  $A$  on a system  $S$ . For the particular case of the system  $L$ , given with a security reduction  $A_L \Rightarrow P \neq NP$ , we first of all assume that we are given an effective algorithm<sup>9</sup> to transform any attack  $A_L$  to a proof of  $P \neq NP$ .

<sup>9</sup>The *constructivist* imperative that an implication like  $A_L \Rightarrow P \neq NP$  should be supported by an effective algo-

Denote the time complexity of the shortest such algorithm by

$$d = \mathcal{D}(A_L, P \neq NP)$$

where we abuse notation and write  $P \neq NP$  for the algorithm that outputs a proof of the statement  $P \neq NP$ . Now if the system  $L$  itself does not allow shortening of the proof  $A_L \implies P \neq NP$ , i.e. if  $d \geq \mathcal{D}(L \wedge A_L, P \neq NP)$ , then we can get

$$\mathcal{D}(L, A_L) + d \geq \mathcal{D}(L, P \neq NP)$$

as a substitution instance of (6). Going back to security by obscurity, this means that, although  $L$  may be vulnerable to a computationally easy attack  $A_L$ , constructing this attack may be logically hard, nearly as hard as deriving a proof of  $P \neq NP$  from it.

**Logical complexity of randomized algorithms.** While logical complexity as a tool and resource for security will clearly be built upon the deep and interesting results of algorithmic information theory, it should be noted that a realistic model of attacker’s logical practices will require algorithmic information theory of *randomized* computation, since the algorithms used in security tend to be randomized. The encoding in (1) will thus not be strict but only approximate, with (2) replaced with

$$\Pr(U(\ulcorner P \urcorner_\varepsilon, \ulcorner Q \urcorner_\varepsilon, x) = P(Q(x))) > \varepsilon \quad (7)$$

### 4.3 Logical complexity of gaming security

Randomized logical complexity brings us back to security as a game of incomplete information. In order to construct a strategy each player in such a game must supplement the available informations about the opponent by some *beliefs*. Mathematically, these beliefs have been modeled, ever since [20], as probability distributions over opponent’s payoff functions. More generally, in games not driven by payoffs, beliefs can be modeled as probability distributions over the possible opponent’s behaviors, or algorithms. In terms of randomized logical complexity, such a belief can thus be viewed as an approximate logical specification of opponent’s algorithms.

Since both players in a game of incomplete information are building beliefs about each other, they must also build beliefs about each other beliefs:  $A$  formulates a belief about  $B$ ’s belief about  $A$ , and  $B$  formulates a belief about  $A$ ’s belief about  $B$ . And so on to the infinity. This is described in more detail in the Appendix, and in still more detail in [4]. These hierarchies of beliefs are formalized as probability distributions over probability distributions. The framework of approximate logical complexity now captures the way in which the players encode their beliefs about each other’s beliefs. This leads into an *algorithmic* theory of incomplete information, where players’ belief hierarchies consist of *sampleable* probability distributions.

---

rithm transforming any proof of the antecedent into a proof of the consequence was goes back to Brouwer [47]. Although constructivism was deemed impractical by most mathematicians, functional programming can be viewed as its practical realization [19, 34, 35].

## 5. FINAL COMMENTS

**On games of security and obscurity.** The first idea of this paper is that security is a game of *incomplete* information: by analyzing your enemy’s behaviors and algorithms (subsumed under what game theorists call his *type*), and by obscuring your own, you can improve the odds of winning this game.

This claim contradicts Kerckhoffs’ Principle that there is no security by obscurity, which implies that security should be viewed as a game of *imperfect* information, by asserting that security is based on players’ secret data (e.g. cards), and not on their obscure behaviors and algorithms.

I described a toy model of a security game which illustrates that security is fundamentally based on gathering and analyzing information about the type of the opponent. This model thus suggests that security is *not* a game of imperfect information, but a game of *incomplete* information. If confirmed, this claim implies that security can be increased not only by analyzing attacker’s type, but also by obscuring defender’s type.

**On logical complexity.** The second idea of this paper is the idea of *one-way programming*, based on the concept of *logical complexity* of programs. The suggestion is that algorithmic information theory may be a useful new area for security research. It provides a natural conceptual framework for studying algorithm evolution driven by the battles between the attackers and defenders. It also provides some concrete technical tools, which raise a possibility of systems vulnerable to computationally feasible, but logically unfeasible attacks; in other words, the attack algorithms that may be easy to run when you know them, but hard to construct if you don’t know them. Security experts dismiss such ideas, mainly because the experience and theory show that algorithms are never too hard to *reconstruct* from their obfuscations. But constructing them from other algorithms can be genuinely hard.

**On security of profiling.** Typing and profiling are frowned upon in security. Leaving aside the question whether gathering information about the attacker, and obscuring the system, might be useful for security or not, these practices remain questionable socially. The false positives arising from such methods cause a lot of trouble, and tend to just drive the attackers deeper into hiding.

On the other hand, typing and profiling are technically and conceptually unavoidable in gaming, and remain respectable research topics of game theory. Some games cannot be played without typing and profiling the opponents. Poker and the bidding phase of bridge are all about trying to guess your opponents’ secrets by analyzing their behaviors. Players do all they can to avoid being analyzed, and many prod their opponents to sample their behaviors. Some games cannot be won by mere uniform distributions, without analyzing opponents’ biases.

Both game theory and immune system teach us that we cannot avoid profiling the enemy. But both the social experience and immune system teach us that we must set the

thresholds high to avoid the false positives that the profiling methods are so prone to. Misidentifying the enemy leads to auto-immune disorders, which can be equally pernicious socially, as they are to our health.

## 6. REFERENCES

- [1] Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. Full completeness for PCF. *Information and Computation*, 163:409–470, 2000.
- [2] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.
- [3] George A. Akerlof. The market for 'lemons': Quality uncertainty and the market mechanism. *Quarterly Journal of Economics*, 84(3):488–500, 1970.
- [4] Robert J. Aumann and Aviad Heifetz. Incomplete information. In Robert J. Aumann and Sergiu Hart, editors, *Handbook of Game Theory, Volume 3*, pages 1665–1686. Elsevier/North Holland, 2002.
- [5] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '01*, pages 1–18, London, UK, 2001. Springer-Verlag.
- [6] Charles H. Bennett. The thermodynamics of computation—a review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.
- [7] Charles H. Bennett. Logical depth and physical complexity. In *A half-century survey on The Universal Turing Machine*, pages 227–257, New York, NY, USA, 1988. Oxford University Press, Inc.
- [8] Elwyn Berlekamp, John Conway, and Richard Guy. *Winning Ways for your Mathematical Plays*, volume 1-2. Academic Press, 1982.
- [9] Dines Bjørner, editor. *Abstract Software Specifications*, volume 86 of *Lecture Notes in Computer Science*. Springer, 1980.
- [10] E. Borger and Robert F. Stark. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
- [11] Gregory Chaitin. Algorithmic information theory. *IBM J. Res. Develop.*, 21:350–359, 496, 1977.
- [12] William Dean. Pitfalls in the use of imperfect information. RAND Corporation Report P-7430, 1988. Santa Monica, CA.
- [13] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
- [14] Lester E. Dubins and Leonard J. Savage. *How to Gamble If You Must*. McGraw-Hill, New York, 1965.
- [15] José Luiz Fiadeiro. *Categories for software engineering*. Springer, 2005.
- [16] Electronic Frontier Foundation. Press releases and court documents on DVD CCA v Bunner Case. [w2.eff.org/IP/Video/DVDCCA\\_case/#bunner-press](http://w2.eff.org/IP/Video/DVDCCA_case/#bunner-press). (Retrieved on July 31, 2011).
- [17] Amanda Friedenberga and Martin Meier. On the relationship between hierarchy and type morphisms. *Economic Theory*, 46:377–399, 2011. 10.1007/s00199-010-0517-2.
- [18] Drew Fudenberg and Jean Tirole. *Game Theory*. MIT Press, August 1991.
- [19] Philippe De Groote. *The Curry-Howard Isomorphism*. Academia, 1995.
- [20] John C Harsanyi. Games with incomplete information played by Bayesian players, I-III. *Management Science*, 14:159–182, 320–334, 486–502, 1968.
- [21] C. A. R. Hoare. Programs are predicates. *Phil. Trans. R. Soc. Lond.*, A 312:475–489, 1984.
- [22] J. Martin E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.
- [23] International Obfuscated C Code Contest. [www0.us.ioccc.org/main.html](http://www0.us.ioccc.org/main.html). (Retrieved on 31 July 2011).
- [24] Auguste Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, IX:5–38, 161–191, 1883.
- [25] Stephen C. Kleene. Realizability: a retrospective survey. In A.R.D. Mathias and H. Rogers, editors, *Cambridge Summer School in Mathematical Logic*, volume 337 of *Lecture Notes in Mathematics*, pages 95–112. Springer-Verlag, 1973.
- [26] Andrey N. Kolmogorov. On tables of random numbers (Reprinted from "Sankhya: The Indian Journal of Statistics", Series A, Vol. 25 Part 4, 1963). *Theor. Comput. Sci.*, 207(2):387–395, 1998.
- [27] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. Foundations of attack-defense trees. In Pierpaolo Degano, Sandro Etalle, and Joshua D. Guttman, editors, *Formal Aspects in Security and Trust*, volume 6561 of *Lecture Notes in Computer Science*, pages 80–95. Springer, 2010.
- [28] Leonid A. Levin. Randomness conservation inequalities: Information and independence in mathematical theories. *Information and Control*, 61:15–37, 1984.
- [29] Ming Li and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications (2. ed.)*. Graduate texts in computer science. Springer, 1997.
- [30] Jean F. Mertens and Shmuel Zamir. Formulation of Bayesian analysis for games with incomplete information. *Internat. J. Game Theory*, 14(1):1–29, 1985.
- [31] Tyler Moore, Allan Friedman, and Ariel D. Procaccia. Would a 'cyber warrior' protect us: exploring trade-offs between attack and defense of information systems. In *Proceedings of the 2010 workshop on New security paradigms, NSPW '10*, pages 85–94, New York, NY, USA, 2010. ACM.
- [32] Till Mossakowski. Relating casl with other specification languages: the institution level. *Theor. Comput. Sci.*, 286(2):367–475, 2002.
- [33] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.
- [34] Dusko Pavlovic. Constructions and predicates. In D. Pitt et al., editor, *Category Theory and Computer Science '91*, volume 530 of *Lecture Notes in Computer Science*, pages 173–197. Springer Verlag, 1991.

- [35] Dusko Pavlovic. Maps II: Chasing diagrams in categorical proof theory. *J. of the IGPL*, 4(2):1–36, 1996.
- [36] Dusko Pavlovic. A semantical approach to equilibria and rationality. In Alexander Kurz and Andrzej Tarlecki, editors, *Proceedings of CALCO 2009*, volume 5728 of *Lecture Notes in Computer Science*, pages 317–334. Springer Verlag, 2009. arxiv.org:0905.3548.
- [37] Dusko Pavlovic and Douglas R. Smith. Guarded transitions in evolving specifications. In H. Kirchner and C. Ringeissen, editors, *Proceedings of AMAST 2002*, volume 2422 of *Lecture Notes in Computer Science*, pages 411–425. Springer Verlag, 2002.
- [38] Dusko Pavlovic and Douglas R. Smith. Software development by refinement. In Bernhard K. Aichernig and Tom Maibaum, editors, *Formal Methods at the Crossroads*, volume 2757 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.
- [39] Duško Pavlović. Semantics of first order parametric specifications. In J. Woodcock and J. Wing, editors, *Formal Methods '99*, volume 1708 of *Lecture Notes in Computer Science*, pages 155–172. Springer Verlag, 1999.
- [40] Claude Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.
- [41] Ray J. Solomonoff. A formal theory of inductive inference. Part I., Part II. *Information and Control*, 7:1–22, 224–254, 1964.
- [42] Michael Spence. Job market signaling. *Quarterly Journal of Economics*, 87(3):355–374, 1973.
- [43] Frank A. Stevenson. Cryptanalysis of Contents Scrambling System. [http://web.archive.org/web/-20000302000206/www.dvd-copy.com/news/cryptanalysis\\_of\\_contents\\_scrambling\\_system.htm](http://web.archive.org/web/-20000302000206/www.dvd-copy.com/news/cryptanalysis_of_contents_scrambling_system.htm), 8 November 1999. (Retrieved on July 31, 2011).
- [44] George J Stigler. The economics of information. *Journal of Political Economy*, 69(3):213–225, 1961.
- [45] B. Stone-Gross, M. Cova, B. Gilbert, R. Kemmerer, C. Kruegel, and G. Vigna. Analysis of a Botnet Takeover. *IEEE Security and Privacy Magazine*, 9(1):64–72, January 2011.
- [46] Brett Stone-Gross, Marco Cova, Bob Gilbert, Lorenzo Cavallaro, Martin Szydlowski, Christopher Kruegel, Giovanni Vigna, and Richard Kemmerer. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proceedings of the Computer and Communications Security Conference (CCS)*, Chicago, IL, November 2009.
- [47] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics: an Introduction*, volume I and II. North-Holland, Amsterdam, 1988.
- [48] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ, 1944.
- [49] A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the algorithmic concepts of information and randomness. *Russian Math. Surveys*, 25(6):83–124, 1970.

## APPENDIX

### A. GAMING SECURITY FORMALISM

Can the idea of applied security by obscurity be realized? To test it, let us first make it more precise in a mathematical model. I first present a very abstract model of strategic behavior, capturing and distinguishing the various families of games studied in game theory, and some families not studied. The model is based on coalgebraic methods, along the lines of [36]. I will try to keep the technicalities at a minimum, and the reader is not expected to know what is a coalgebra.

#### A.1 Arenas

*Definition 1.* A *player* is a pair of sets  $A = \langle M_A, S_A \rangle$ , where the elements of  $M_A$  represent or *moves* available to  $A$ , and the elements of  $S_A$  are the *states* that  $A$  may observe.

A *simple response*  $\Sigma : A \rightarrow B$  for a player  $B$  to a player  $A$  is a binary relation

$$\Sigma : M_A \times S_B^2 \times M_B \rightarrow \{0, 1\}$$

When  $\Sigma(a, \beta, \beta', b) = 1$ , we write  $\langle a, \beta \rangle \xrightarrow{\Sigma} \langle \beta', b \rangle$ , and say that the strategy  $\Sigma$  at  $B$ 's state  $\beta$  prescribes that  $B$  should respond to  $A$ 's move  $a$  by the move  $b$  and update his state to  $\beta'$ . The set of  $B$ 's simple responses to  $A$  is written  $\text{SR}(A, B)$ .

A *mixed response*  $\Phi : A \rightarrow B$  for the player  $B$  to the player  $A$  is a matrix

$$\Phi : M_A \times S_B^2 \times M_B \rightarrow [0, 1]$$

required to be finitely supported and stochastic in  $M_A$ , i.e. for every  $a \in M_A$  holds

- $\Phi_{a\beta\beta'b} = 0$  holds for all but finitely many  $\beta, \beta'$  and  $b$ ,
- $\sum_{\beta\beta'b} \Phi_{a\beta\beta'b} = 1$ .

When  $\Phi_{a\beta\beta'b} = p$  we write  $\langle a, \beta \rangle \xrightarrow[p]{\Phi} \langle \beta', b \rangle$ , and say that the strategy  $\Phi$  at  $B$ 's state  $\beta$  responds to  $A$ 's move  $a$  with a probability  $p$  by  $B$ 's move  $b$  leading him into the state  $\beta'$ . The set of  $B$ 's mixed responses to  $A$  is written  $\text{MR}(A, B)$ .

An *arena* is a specification of a set of players and a set of responses between them.

**Responses compose.** Given simple responses  $\Sigma : A \rightarrow B$  and  $\Gamma : B \rightarrow C$ , we can derive a response  $(\Sigma; \Gamma) : A \rightarrow C$  for the player  $C$  against  $A$  by taking the player  $B$  as a "man in the middle". The derived response is constructed as follows:

$$\frac{\langle a, \beta \rangle \xrightarrow{\Sigma} \langle \beta', b \rangle \quad \langle b, \gamma \rangle \xrightarrow{\Gamma} \langle \gamma', c \rangle}{\langle a, \gamma \rangle \xrightarrow{(\Sigma; \Gamma)} \langle \gamma', c \rangle}$$

Following the same idea, for the mixed responses  $\Phi : A \rightarrow B$  and  $\Psi : B \rightarrow C$  we have the composite  $(\Phi; \Psi) : A \rightarrow C$  with the entries

$$(\Phi; \Psi)_{a\gamma\gamma'c} = \sum_{\beta\beta'b} \Phi_{a\beta\beta'b} \cdot \Psi_{b\gamma\gamma'c}$$

It is easy to see that these composition operations are associative and unitary, both for the simple and for the mixed responses.

## A.2 Games

Arenas turn out to provide a convenient framework for a unified presentation of games studied in game theory [48, 33], mathematical games [8], game semantics [1, 22], and some constructions in-between these areas [14]. Here we shall use them to succinctly distinguish between the various kinds of game with respect to the information available to the players. As mentioned before, game theorists usually distinguish two kinds of players' information:

- data, or positions: e.g., a hand of cards, or a secret number; and
- types, or preferences: e.g., player's payoff matrix, or a system that he uses are components of his type.

The games in which the players have private data or positions are the games of *imperfect information*. The games where the players have private types or preferences, e.g. because they don't know each other's payoff matrices, are the games of *incomplete information*. See [4] for more about these ideas, [33] for the technical details of the perfect-imperfect distinction, and [18] for the technical details of the complete-incomplete distinction.

But let us see how arenas capture these distinction, and what does all that have to do with security.

### A.2.1 Games of perfect and complete information

In games of perfect and complete information, each player has all information about the other player's data and preferences, i.e. payoffs. To present the usual stateless games in normal form, we consider the players  $A$  and  $B$  whose state spaces are the sets payoff bimatrixes, i.e.

$$S_A = S_B = (\mathbb{R} \times \mathbb{R})^{M_A \times M_B}$$

In other words, a state  $\sigma \in S_A = S_B$  is a pair of maps  $\sigma = \langle \sigma^A, \sigma^B \rangle$  where  $\sigma^A$  is the  $M_A \times M_B$ -matrix of  $A$ 's payoffs: the entry  $\sigma_{ab}^A$  is  $A$ 's payoff if  $A$  plays  $a$  and  $B$  plays  $b$ . Ditto for  $\sigma^B$ . Each game in the standard bimatrix form corresponds to one element of both state spaces  $S_A = S_B$ . It is nominally represented as a state, but this state does not change. The main point here is that both  $A$  and  $B$  know this element. This allows both of them to determine the best response strategies  $\Sigma_A : B \rightarrow A$  for  $A$  and  $\Sigma_B : A \rightarrow B$  for  $B$ , in the form

$$\begin{aligned} \langle b, \sigma^A \rangle &\xrightarrow{\Sigma_A} \langle \sigma^A, a \rangle \iff \forall x \in M_A. \sigma_{xb}^A \leq \sigma_{ax}^A \\ \langle a, \sigma^B \rangle &\xrightarrow{\Sigma_B} \langle \sigma^B, b \rangle \iff \forall y \in M_B. \sigma_{ay}^B \leq \sigma_{by}^B \end{aligned}$$

and to compute the Nash equilibria as the fixed points of the composites  $(\Sigma^B; \Sigma^A) : A \rightarrow A$  and  $(\Sigma^A; \Sigma^B) : B \rightarrow B$ . This is further discussed in [36]. Although the payoff matrices in games studied in game theory usually do not change, so the corresponding responses fix all states, and each response actually presents a method to respond in a whole family of games, represented by the whole space of

payoff matrices, it is interesting to consider, e.g. discounted payoffs in some iterated games, where the full force of the response formalism over the above state spaces is used.

### A.2.2 Games of imperfect information

Games of imperfect information are usually viewed in extended form, i.e. with nontrivial state changes, because players' private data can then be presented as their private states. Each player now has a set of private positions,  $P_A$  and  $P_B$ , which is not visible to the opponent. On the other hand, both player's types, presented as their payoff matrices, are still visible to both. So we have

$$\begin{aligned} S_A &= P_A \times (\mathbb{R} \times \mathbb{R})^{M_A \times M_B} \\ S_B &= P_B \times (\mathbb{R} \times \mathbb{R})^{M_A \times M_B} \end{aligned}$$

E.g., in a game of cards,  $A$ 's hand will be an element of  $P_A$ ,  $B$ 's hand will be an element of  $P_B$ . With each response, each player updates his position, whereas their payoff matrices usually do not change.

### A.2.3 Games of incomplete information

Games of incomplete information are studied in *epistemic* game theory [20, 30, 4], which is formalized through knowledge and belief logics. The reason is that each player here only *knows* with certainty his own preferences, as expressed by his payoff matrix. The opponent's preferences and payoffs are kept in obscurity. In order to anticipate opponent's behaviors, each player must build some *beliefs* about the other player's preferences. In the first instance, this is expressed as a probability distribution over the other player's possible payoff matrices. However, the other player also builds beliefs about his opponent's preferences, and his behavior is therefore not entirely determined by his own preferences, but also by his beliefs about his opponent's preferences. So each player also builds some beliefs about the other player's beliefs, which is expressed as a probability distribution over the probability distributions over the payoff matrices. And so to the infinity. Harsanyi formalized the notion of players *type* as an element of such information space, which includes each player's payoffs, his beliefs about the other player's payoffs, his beliefs about the other player's beliefs, and so on [20]. Harsanyi's form of games of incomplete information can be presented in the arena framework by taking

$$\begin{aligned} S_A &= \mathbb{R}^{M_A \times M_B} + \Delta S_B \\ S_B &= \mathbb{R}^{M_A \times M_B} + \Delta S_A \end{aligned}$$

where  $+$  denotes the disjoint union of sets, and  $\Delta X$  is the space of finitely supported probability distributions over  $X$ , which consists of the maps  $p : X \rightarrow [0, 1]$  such that

$$|\{x \in X | p(x) > 0\}| < \infty \quad \text{and} \quad \sum_{x \in X} p(x) = 1$$

Resolving the above inductive definitions of  $S_A$  and  $S_B$ , we get

$$S_A = S_B = \prod_{i=0}^{\infty} \Delta^i (\mathbb{R}^{M_A \times M_B})$$

Here the state  $\sigma^A \in S_A$  is thus a sequence

$$\sigma^A = \langle \sigma_0^A, \sigma_1^A, \sigma_2^A, \dots \rangle$$

where  $\sigma_i^A \in \Delta^i \mathbb{R}^{M_A \times M_B}$ . The even components  $\sigma_{2i}^A$  represent  $A$ 's payoff matrix,  $A$ 's belief about  $B$ 's belief about  $A$ 's payoff matrix,  $A$ 's belief about  $B$ 's belief about  $A$ 's belief about  $B$ 's belief about  $A$ 's payoff matrix, and so on. The odd components  $\sigma_{2i+1}^A$  represent  $A$ 's belief about  $B$ 's payoff matrix,  $A$ 's belief about  $B$ 's belief about  $A$ 's belief about  $B$ 's payoff matrix, and so on. The meanings of the components of the state  $\sigma^B \in S_B$  are analogous.

### A.3 Security games

We model security processes as a special family of games. It will be a game of imperfect information, since the players of security games usually have some secret keys, which are presented as the elements of their private state sets  $P_A$  and  $P_D$ . The player  $A$  is now the attacker, and the player  $D$  is the defender.

The goal of a security game is not expressed through payoffs, but through "security requirements"  $\Theta \subseteq P_D$ . The intuition is that the defender  $D$  is given a family of assets to protect, and the  $\Theta$  are the desired states, where these assets are protected. The defender's goal is to keep the state of the game in  $\Theta$ , whereas the attacker's goal is to drive the game outside  $\Theta$ . The attacker may have additional preferences, expressed by a probability distribution over his own private states  $P_A$ . We shall ignore this aspect, since it plays no role in the argument here; but it can easily be captured in the arena formalism.

Since players' goals are not to maximize their revenues, their behaviors are not determined by payoff matrices, but by their response strategies, which we collect in the sets  $\text{RA}(A, D)$  and  $\text{RA}(D, A)$ . In the simplest case, response strategies boil down to the response maps, and we take  $\text{RA}(A, D) = \text{SR}(A, D)$ , or  $\text{RA}(A, D) = \text{MR}(A, D)$ . In general, though,  $A$ 's and  $D$ 's behavior may not be purely extensional, and the elements of  $\text{RA}(A, D)$  may be actual algorithms.

While both players surely keep their keys secret, and some part of the spaces  $P_A$  and  $P_D$  are private, they may not know each other's preferences, and may not be given each other's "response algorithms". If they do know them, then both defender's defenses and attacker's attacks are achieved without obscurity. However, modern security definitions usually require that the defender defends the system against a family of attacks without querying the attacker about his algorithms. So at least the *theoretical attacks are in principle afforded the cloak of obscurity*. Since the defender  $D$  thus does not know the attacker  $A$ 's algorithms, we model security games as games of incomplete information, replacing the player's spaces of payoff matrices by the spaces  $\text{RA}(A, D)$  and  $\text{RA}(D, A)$  of their response strategies to one another.

Like above,  $A$  thus only knows  $P_A$  and  $\text{RA}(D, A)$  with certainty, and  $D$  only knows  $P_D$  and  $\text{RA}(A, D)$  with certainty. Moreover,  $A$  builds his beliefs about  $D$ 's data and type, as a probability distribution over  $P_D \times \text{RA}(A, D)$ , and  $D$  builds similar beliefs about  $A$ . Since they then also have to build beliefs about each other's beliefs, we have a mutually recurrent definition of the state spaces again:

$$\begin{aligned} S_A &= (P_A \times \text{RA}(D, A)) + \Delta S_D \\ S_D &= (P_D \times \text{RA}(A, D)) + \Delta S_A \end{aligned}$$

Resolving the induction again, we now get

$$\begin{aligned} S_A &= \prod_{i=0}^{\infty} \Delta^{2i} (P_A \times \text{RA}(D, A)) \times \Delta^{2i+1} (P_D \times \text{RA}(A, D)) \\ S_D &= \prod_{i=0}^{\infty} \Delta^{2i} (P_D \times \text{RA}(A, D)) \times \Delta^{2i+1} (P_A \times \text{RA}(D, A)) \end{aligned}$$

Defender's state  $\beta \in S_D$  is thus a sequence  $\beta = \langle \beta_0, \beta_1, \beta_2, \dots \rangle$ , where

- $\beta_0 = \langle \beta_0^P, \beta_0^{\text{RA}} \rangle \in P_D \times \text{RA}(A, D)$  consists of
  - $D$ 's secrets  $\beta_0^P \in P_D$ , and
  - $D$ 's current response strategy  $\beta_0^{\text{RA}} \in \text{RA}(A, D)$
- $\beta_1 \in \Delta(P_A \times \text{RA}(D, A))$  is  $D$ 's belief about  $A$ 's secrets and her response strategy;
- $\beta_2 \in \Delta^2(P_D \times \text{RA}(A, D))$  is  $D$ 's belief about  $A$ 's belief about  $D$ 's secrets and his response strategy;
- $\beta_3 \in \Delta^3(P_A \times \text{RA}(D, A))$  is  $D$ 's belief about  $A$ 's belief about  $D$ 's beliefs, etc.

Each response strategy  $\Sigma : A \rightarrow D$  prescribes the way in which  $D$  should update his state in response to  $A$ 's observed moves. E.g., if  $\text{RA}(D, A)$  is taken to consist of relations in the form  $\Lambda : M_D^+ \times M_A \rightarrow \{0, 1\}$ , where  $M_D^+$  is the set of nonempty strings in  $M_D$ , then  $D$  can record the longer and longer histories of  $A$ 's responses to his moves.

**Remark.** The fact that, in a security game,  $A$ 's state space  $S_A$  contains  $\text{RA}(D, A)$  and  $\text{RA}(A, D)$  means that each player  $A$  is prepared for playing against a particular player  $D$ ; while  $D$  is prepared for playing against  $A$ . This reflects the situation in which all security measures are introduced with particular attackers in mind, whereas the attacks are built to attack particular security measures. A technical consequence is that players' state spaces are defined by the inductive clauses, which often lead to complex impredicative structures. This should not be surprising, since even informal security considerations often give rise to complex belief hierarchies, and the formal constructions of epistemic game theory [20, 30, 17] seem like a natural tool to apply.

### A.4 The game of attack vectors

We specify a formal model of the game of attack vectors as a game of perfect but incomplete information. This means that the players know each other's positions, but need to learn about each other's type, i.e. plans and methods. The assumption that the players know each other's position could be removed, without changing the outcomes and strategies, by refining the way the model of players' observations. But this seems inessential, and we omit it for simplicity.

Let  $\mathcal{O}$  denote the unit disk in the real plane. If we assume that it is parametrized in the polar coordinates, then  $\mathcal{O} = \{0\} \cup (0, 1] \times \mathbb{R}/2\pi\mathbb{Z}$ , where  $\mathbb{R}/2\pi\mathbb{Z}$  denotes the circle. Let  $\mathcal{R} \subseteq \mathbb{R}^2$  be an open domain in the real plane. Players' positions can then be defined as continuous mappings of  $\mathcal{O}$  into  $\mathcal{R}$ , i.e.

$$P_A = P_D = \mathcal{R}^{\mathcal{O}}$$



The rules of the game will be such that the attacker's and the defender's positions  $p_A, p_D \in \mathcal{R}^\mathcal{O}$  always satisfy the constraint that

$$p_A^\circ \cap p_D^\circ = \emptyset$$

where  $p^\circ$  denotes the interior of the image of  $p : \mathcal{O} \rightarrow \mathcal{R}$ . The assumption that both players know both their own and the opponent's positions means that both state spaces  $S_A$  and  $S_D$  will contain  $P_A \times P_D$  as a component. The state spaces are thus

$$\begin{aligned} S_A &= (P_A \times P_D \times \text{RA}(D, A)) + \Delta S_D \\ S_D &= (P_A \times P_D \times \text{RA}(A, D)) + \Delta S_D \end{aligned}$$

To start off the game, we assume that the defender  $D$  is given some assets to secure, presented as an area  $\Theta \subseteq \mathcal{R}$ . The defender wins as long as his position  $p_D \in P_D$  is such that  $\Theta \subseteq p_D^\circ$ . Otherwise the defender loses. The attacker's goal is to acquire the assets, i.e. to maximize the area  $\Theta \cap p_A^\circ$ .

The players are given equal forces to distribute along the borders of their respective territories. Their moves are the choices of these distributions, i.e.

$$M_A = M_D = \Delta(\partial\mathcal{O})$$

where  $\partial\mathcal{O}$  is the unit circle, viewed as the boundary of  $\mathcal{O}$ , and  $\Delta(\partial\mathcal{O})$  denotes the distributions along  $\partial\mathcal{O}$ , i.e. the measurable functions  $m : \partial\mathcal{O} \rightarrow [0, 1]$  such that  $\int_{\partial\mathcal{O}} m = 1$ .

How will  $D$  update his space after a move? This can be specified as a simple response  $\Sigma : A \rightarrow D$ . Since the point of this game is to illustrate the need for learning about the opponent, let us leave out players' type information for the moment, and assume that the players only look at their positions, i.e.  $S_A = S_D = P_A \times P_D$ . To specify  $\Sigma : A \rightarrow B$ , we must thus determine the relation

$$\langle m_A, p_A, p_D \rangle \xrightarrow{\Sigma} \langle p'_A, p'_D, m_D \rangle$$

for any given  $m_A \in M_A$ ,  $p_A \in P_A$ , and  $p_D \in P_D$ . We describe the updates  $p'_A$  and  $p'_D$  for an arbitrary  $m_D$ , and leave it to  $D$  to determine which  $m_D$ s are the best responses for him. So given the previous positions and both player's moves, the new position  $p'_A$  will map a point on the boundary of the circle, viewed as a unit vector  $\vec{x} \in \mathcal{O}$  into the vector  $\vec{p}'_A(\vec{x})$  in  $\mathcal{R} \subseteq \mathbb{R}^2$  as follows.

- If for all  $\vec{y} \in \mathcal{O}$  and for all  $s \in [0, m_A(\vec{x})]$  and all  $t \in [0, m_D(\vec{y})]$  holds  $(1+s)\vec{p}_A(\vec{x}) \neq (1+t)\vec{p}_D(\vec{y})$  then set

$$\vec{p}'_A(\vec{x}) = \frac{1+m_A(\vec{x})}{2} \cdot \vec{p}_A(\vec{x})$$

- Otherwise, let  $\vec{y} \in \mathcal{O}$ ,  $s \in [0, m_A(\vec{x})]$  and  $t \in [0, m_D(\vec{y})]$  be the smallest numbers such that  $(1+s)\vec{p}_A(\vec{x}) = (1+t)\vec{p}_D(\vec{y})$ . Writing  $m'_A(\vec{x}) = m_A(\vec{x}) - s$  and  $m'_D(\vec{y}) = m_D(\vec{y}) - t$ , we set

$$\vec{p}'_A(\vec{x}) = \frac{1+m'_A(\vec{x})}{2} \cdot \vec{p}_A(\vec{x}) + \frac{1+m'_D(\vec{y})}{2} \cdot \vec{p}_D(\vec{y})$$

This means that the player  $A$  will push her boundary by  $m_A(\vec{x})$  in the direction  $\vec{p}_A(\vec{x})$  if she does not encounter  $D$  at any point during that push. If somewhere during that push she does encounter  $D$ 's territory, then they will push against each other, i.e. their push vectors will compose. More precisely,  $A$  will push in the direction  $\vec{p}_A(\vec{x})$  with the force  $m'_A(\vec{x}) = m_A(\vec{x}) - s$ , that remains to her after the initial free push by  $s$ ; but moreover, her boundary will also be pushed in the direction  $\vec{p}_D(\vec{y})$  by the boundary of  $D$ 's territory, with the force  $m'_D(\vec{y}) = m_D(\vec{y}) - t$ , that remains to  $D$  after his initial free push by  $t$ . Since  $D$ 's update is defined analogously, a common boundary point will arise, i.e. players' borders will remain adjacent. When the move next, there will be no free initial pushes, i.e.  $s$  and  $t$  will be 0, and the update vectors will compose in full force.

How do the players compute the best moves? Attacker's goal is, of course, to form a common boundary and to push towards  $\Theta$ , preferably from the direction where the defender does not defend. The defender's goal is to push back. As explained in the text, the game is thus resolved on defender's capability to predict attacker's moves. Since the territories do not intersect, but  $A$ 's moves become observable for  $D$  along the part of the boundary of  $A$ 's territory that lies within the convex hull of  $D$ 's territory,  $D$ 's moves must be selected to maximize the length of the curve

$$\partial p_A \cap \text{conv}(p_D)$$

This strategic goal leads to the evolution described informally in Sec. 3.2.